ActiveX Controls and Dialog Boxes

Contents

Microsoft **Excel** 97, Word 97, and PowerPoint 97 share powerful new tools for creating custom dialog boxes. Because these applications use the same dialog b tools in the Visual Basic Editor, you only have to learn how to create custom dia boxes one way for all three applications, and you can share these dialog boxes across applications.

After you've created a custom dialog box, you can add **ActiveX** controls (previo referred to as OLE controls) to it. You can also place **ActiveX** controls directly o document, worksheet, or slide. To determine how custom dialog boxes and cont respond to specific user actions — for example, clicking a control or changing its value — you write event procedures that will run whenever a specific event occu

Note For information about designing forms in Microsoft Access, see *Building Applications with Microsoft Access 97*, available in Microsoft Access 97 and Micr Office 97, Developer Edition. An online version of *Building Applications with Microsoft Access 97* is available in the Value Pack on CDROM in Microsoft Access and Microsoft Office 97, Professional Edition. For information about designing fo in Microsoft Outlook 97, see *Building Microsoft Outlook 97 Applications* by Peter Krebs, available from Microsoft Press.

Designing Custom Dialog Boxes

To create a custom dialog box, you must create a form (also called a *UserForm*) contain controls, add controls to the form, set properties for the controls, and w code that responds to form and control events.

Note When you're in the Visual Basic Editor designing your dialog box, you're design mode. In design mode, you can edit controls. Controls don't respond to events in design mode. When you run your dialog box — that is, when you displ it the way users will see it — you're in run mode. Controls do respond to events run mode.

Creating a New Dialog Box

Every custom dialog box in your project is a form, or UserForm. A new UserForr contains a title bar and an empty area in which you can place controls.

To create a new UserForm

- On the Insert menu in the Visual Basic Editor, click UserForm.

A new, empty UserForm is displayed. Use the Properties window to set properti for the UserForm — that is, to change the name, behavior, and appearance of tl form. For example, to change the caption on a form, set the Caption property. F more information about the Properties window and the Visual Basic Editor, see Chapter 1, "Programming Basics."

Adding Controls to a Custom Dialog Box

Use the **Toolbox** to add controls to a custom dialog box. Click **Toolbox** on the **View** menu to display the **Toolbox** if it's not already visible. To see the name o particular control in the **Toolbox**, position the mouse pointer over that control.

To add a control to a custom dialog box, find the control you want to add in the **Toolbox**, drag the control onto the form, and then drag one or more of the control's adjustment handles until the control is the size and shape you want. F more information about a specific type of control, add the control to a form, sel the control, and then press F1.

Note Dragging a control (or a number of "grouped" controls) from a custom d box back to the Toolbox creates a template of that control, which you can reuse any time. This is a useful feature for implementing a standard "look and feel" fo your applications.

After you've added controls to the form, use the commands on the **Format** mer or the buttons on the **UserForm** toolbar in the Visual Basic Editor to adjust the alignment and spacing of the controls. Use the **Tab Order** dialog box (**View** me to set the tab order of the controls on the form.

Tip The Visual Basic Editor sets the **TabIndex** property for the controls to determine the tab order. If you want to prevent users from tabbing to a particul control, you can set the **TabStop** property to **False** for that control. To do this, rightclick the control, and then click **Properties** to display the Properties windo

▸Practice 1: Design and run a custom dialog box

1. Create a new UserForm.
2. On the UserForm, insert a **Frame** control.
3. In the **Frame** control, insert three **OptionButton** controls.
4. Click **Run Sub/UserForm** on the **Run** menu.

   The custom dialog box is displayed. The option buttons should work when click them. Because you first created a Frame control to contain the optio buttons, clicking one option button automatically turns all the other ones in that control.
5. Click the **Close** button on the UserForm title bar to exit run mode and ret to design mode.

Setting Control and Dialog Box Properties at Design Time

You can set some control properties at design time (before any macros are run) design mode, rightclick a control and then click **Properties** on the shortcut mer display the **Properties** window. Property names are listed in the lefthand colur the window, and property values are listed in the righthand column. You set a property value by typing the new value in the space to the right of the property name.

Tip You can view the properties of an object either sorted alphabetically (on th **Alphabetic** tab in the **Properties** window) or sorted into functional categories the **Categorized** tab).

▸Practice 2: Set control properties in design mode

1. Create a new UserForm.
2. Add an **Image** control, a **CommandButton** control, and a few other cont to the UserForm.
3. Rightclick the image you added, click **Properties** on the shortcut menu tc display the **Properties** window for the image, and then find **Picture** (for **Picture** property) in the list of properties. To browse for files that you car this property to, click the ellipsis button (...) in the space to the right of **Picture**. Select a file in the **Load Picture** dialog box, and then click **OK**.
4. Click the **CommandButton** you added; the list of properties in the **Properties** window changes to the properties of command buttons. Find **Caption** and type **Send Order** in the space to the right to set the value o the **Caption** property. The caption is the text that appears on the face of command button.

5. In the list of properties for the command button, type **CmdSendOrder** in space to the right of **(Name)**. This sets the name you use to refer to the button in your code.

6. In the list of properties for the command button, type **Click here to senc order** in the space to the right of **ControlTipText**. When the user positio the mouse pointer over this command button in run mode, this tip will appear, indicating what the button does.

7. In the list of properties for the command button, type **s** in the space to th right of **Accelerator**. Notice that the "S" in the "Send Order" caption on t command button is now underlined. (If you choose as an accelerator key letter that isn't in the control caption, there will by no visual indication tha the control has an accelerator key.) While any dialog box is running, the u can press ALT+the accelerator key (in this case, "S") to move the focus directly to the control.

8. On the **Run** menu, click **Run Sub/UserForm**, and then move the focus t control other than the **Send Order** button. You can press ALT+S to move focus to the **Send Order** button.

9. Click the **Close** button on the UserForm title bar to exit run mode and ret to design mode.

Tip To set a property for several controls at the same time, select the controls then change the value for that property in the **Properties** window.

▸Practice 3: Set UserForm properties in design mode

1. Click anywhere in a UserForm except on a control to select the UserForm.

2. In the **Properties** window, type **Book Order Form** in the space to the ri of **Caption**.

3. In the space to the right of **BackColor** in the list of properties, click the a to see a set of values from which to choose. Click the **Palette** tab, and th click the color you want to set as the background color for the dialog box.

4. To see the results of your new settings, run the dialog box. Click the **Clos** button on the title bar to return to design mode.

Creating Tabs in Dialog Boxes

If you need for a single dialog box to handle a large number of controls that car sorted into categories, you can create a dialog box with two or more tabs and tl place different sets of related controls on different tabs in the dialog box. To cre a dialog box with tabs, add a **MultiPage** control to the dialog box and then add controls to each tab (or *page*). To add, remove, rename, or move a page in a **MultiPage** control, rightclick one of the pages in design mode, and then click a command on the shortcut menu.

Note Don't confuse MultiPage controls with TabStrip controls. The pages (or ta of a MultiPage control contain a unique set of controls that you add during desig time to each page. Using a TabStrip control, which can look like a series of tabs buttons, you can modify the values of a shared set of controls at run time. For information about using TabStrip controls, see "Displaying a Custom Dialog Box later in this chapter.

Writing Code to Respond to Dialog Box and Control Events

Each form or control recognizes a predefined set of events, which can be trigger either by the user or by the system. For example, a command button recognize: the Click event that occurs when the user clicks that button, and a form recogni the Initialize event that occurs when the form is loaded. To specify how a form (control should respond to events, you write *event procedures.*

To write an event procedure for a form or control, open the **Code** window by doubleclicking the UserForm or control object, and then click the event name in **Procedure** box (in the upperright corner of the window. Event procedures inclu the name of the UserForm or control. For example, the name of the Click event procedure for the command button Command1 is Command1_Click. For more information about writing event procedures, see Chapter 1, "Programming Basic

‣Practice 4: Write and run an event procedure for a command button

1. Create a new UserForm, and then add a **CommandButton**, a **CheckBox**, and a **ComboBox** control to it.

2. Click the command button. In the **Properties** window, change the code n of the command button by typing **CmdSendOrder** in the space to the rig **(Name)**.

3. Doubleclick the command button to view the code associated with it. By default, the Click event procedure will be displayed in the **Code** window.

4. Add a statement to the CmdSendOrder_Click procedure to display a simpl message box (use the following example).

```
Private Sub CommandButton1_Click()
    MsgBox "I've been clicked once"
End Sub
```

5. Run the dialog box to see the results. The CmdSendOrder_Click event procedure will run every time this command button is clicked in run mode Because you haven't written code for the other controls yet, they don't respond to your mouse actions. Click the **Close** button on the title bar to return to design mode.

To see all the events that command buttons recognize, click the down arrow ne: the **Procedure** box in the **Code** window. Events that already have procedures written for them appear bold. Click an event name in the list to display its associated procedure.

To see the events for a different control on the same UserForm or for the UserF
itself, click the object name in the **Object** box in the **Code** window, and then cli
the arrow next to the **Procedure** box.

Tip If you add code to an event procedure before you change the code name c
the control, your code will still have its previous code name in any procedures it
used in. For example, assume that you add code to the Click event for the
Command1 button and then rename the control as Command2. When you doub
click Command2, you won't see any code in the Click event procedure; You'll ne
to move code from Command1_Click to Command2_Click. To simplify developm
it's a good idea to name your controls with the names you really want for them
before you write any code.

Using Custom Dialog Boxes

To exchange information with the user by way of a custom dialog box, you mus
display the dialog box to the user, respond to user actions in the dialog box, an
when the dialog box is closed, get information that the user entered in it.

Displaying a Custom Dialog Box

When you want to display a custom dialog box to yourself for testing purposes,
click **Run Sub/UserForm** on the **Run** menu in the Visual Basic Editor. Howeve
when you want to display a dialog box to a user, you use the **Show** method. Th
following example displays the dialog box named "UserForm1."

```
UserForm1.Show
```

Getting and Setting Properties at Run Time

If you want to set default values for controls in a custom dialog box, modify
controls while the dialog box is visible, and have access to the information that
user enters in the dialog box, you must set and read the values of control
properties at run time.

Setting Initial Values for Controls

To set the initial value, or *default value*, that a control will have every time the
dialog box that contains it is displayed, add code to the Initialize event procedu
for the UserForm that contains the control that sets the value for the control. W
you display the dialog box, the Initialize event will be triggered, and the control'
value will be initialized.

▸Practice 5: Write and run an Initialize event procedure for a UserForm

1. Create a new UserForm, and then add a **TextBox**, a **ListBox**, and a **CheckBox** control to it.
2. Click the text box. In the **Properties** window, type **txtCustomerName** ir
space to the right of **(Name)** to set the code name of the text box. Then

change the code name of the list box to "lstRegions," change the code nai of the check box to "chkSendExpress," and change the code name of the UserForm itself to "frmPhoneOrders."

3. Doubleclick the UserForm to display the **Code** window. With **UserForm** selected in the **Object** box of the **Code** window, select **Initialize** in the **Procedure** box. Complete the UserForm_Initialize procedure, as shown ir following example.

```
Private Sub UserForm_Initialize()
    With frmPhoneOrders
       .txtCustomerName.Text = "Grant Clarridge"   'Sets default text
       .chkSendExpress.Value = True   'Checks check box by default
       With .lstRegions
            .AddItem "North"         'These lines populate the list box
            .AddItem "South"
            .AddItem "East"
            .AddItem "West"
            .ListIndex = 3           'Selects the 4th item in the list
       End With
    End With
End Sub
```

Note Although collections in the Microsoft **Excel**, Word, and PowerPoint object models are 1based, arrays and collections associated with forms ar based. Therefore, to select the fourth item in the list in the preceding example, you must set the ListIndex property to 3.

4. Run the dialog box to see the results. Click the **Close** button on the title b to return to design mode.

Use Me to Simplify Event Procedure Code

In the preceding example, you can use the keyword **Me** instead of the code name of the UserForm. That is, you can replace the statement `With frmPhoneOrders` with the statement `With Me`. The **Me** keyword used in code for a UserForm or a control on the UserForm represents the UserForm itself. This technique lets you use long, meaningful names for controls while still making code easy to write. Many examples in this chapter demonstrate how to use **Me** this way.

If you want to set the initial value (default value) for a control but you don't wa that to be the initial value every time you call the dialog box, you can use Visua Basic code to set the control's value before you display the dialog box that cont the control. The following example uses the **AddItem** method to add data to a box, sets the value of a text box, and displays the dialog box that contains thes controls.

```
Private Sub GetUserName()
          With UserForm1
                    .lstRegions.AddItem "North"
                    .lstRegions.AddItem "South"
                    .lstRegions.AddItem "East"
                    .lstRegions.AddItem "West"
                    .txtSalesPersonID.Text = "00000"
                    .Show
```

```
                 ' ...
          End With
End Sub
```

Setting Values to Modify Controls While a Dialog Box Is Running

You can set properties and apply methods of controls and the UserForm while a dialog box is running. The following example sets the text (the **Text** property) c TextBox1 to "Hello."

```
TextBox1.Text = "Hello"
```

By setting control properties and applying control methods at run time, you can make changes in a running dialog box in response to a choice the user makes. F example, if you want a particular control to be available only while a particular check box is selected, you can write code that enables the control whenever the user selects the check box and disables it whenever the user clears the check b

Enabling a Control

You can use the **Enabled** property of a control to prevent the user from making changes to the control unless a specified condition is met. For example, a comm use of the **Enabled** property is in an event procedure for a text box that enable the **OK** button only when the user has entered a value that conforms to a stand pattern.

Setting the **Enabled** property is often used to make a set of option buttons available only while the user has a particular check box selected, as demonstrat in the following practice. This code is included in the Change event procedure fo the check box, and it runs whenever the state of the check box (checked or cleared) changes.

▸Practice 6: **Enable** and disable controls during run time

1. Create a new UserForm, and then add a **CheckBox** control to it. Add a **Frame** control to the UserForm, and then place three **OptionButton** cont within the frame.

2. Doubleclick the check box to switch to the **Code** window. With **CheckBox** selected in the **Object** box in the **Code** window, click **Change** in the **Procedure** box. Complete the CheckBox1_Change procedure as shown in following example.

```
Private Sub CheckBox1_Change()
    With Me
       If .CheckBox1.Value = True Then
            .OptionButton1.Enabled = False
            .OptionButton2.Enabled = False
            .OptionButton3.Enabled = False
       Else
            .OptionButton1.Enabled = True
            .OptionButton2.Enabled = True
            .OptionButton3.Enabled = True
       End If
```

```
        End With
    End Sub
```

3. Run the dialog box; select and clear the check box to see how changing tl state of the check box enables or disables the three option buttons. Click · **Close** button on the title bar to return to design mode.

Setting the Focus to a Control

You can set the focus to a control in a dialog box by using the **SetFocus** metho that control (the control with the focus is the one that responds to keyboard inp from the user).

▸**Practice 7: Set the control focus during run time**

1. Create a new UserForm, and then add a **CheckBox**, an **Image**, and a fev other controls to it. In the **Properties** window, set the **Picture** property c the image to display a graphic.

2. Doubleclick the image to switch to the **Code** window. With **Image1** selecl in the **Object** box in the **Code** window, select **Click** in the **Procedure** bo: Complete the Image1_Click procedure as shown in the following example.

```
Private Sub Image1_Click()
    Me.CheckBox1.SetFocus
End Sub
```

3. Run the dialog box. Give the focus to a control other than CheckBox1. Wh you click Image1, CheckBox1 is given the focus (a dotted rectangle surrou the check box, and you can press the SPACEBAR to select or clear the che box). Click the **Close** button on the title bar to return to design mode.

Displaying and Hiding Parts of a Dialog Box

You can set properties or apply methods of the UserForm itself while a dialog bc running. A common use for this is to expand a UserForm to reveal additional options when the user clicks a command button.

▸**Practice 8: Resize a UserForm during run time**

1. Create a new UserForm. The value of its **Height** property (the number to right of **Height** in the **Properties** window) should be 180.

2. Add a **CommandButton** control at the top of the UserForm, and then adc **CheckBox** control to the bottom of the UserForm (the **Top** property for tl check box should be at least 120).

3. Doubleclick the UserForm to switch to the **Code** window. With **UserForm** selected in the **Object** box of the **Code** window, click **Initialize** in the **Procedure** box. Complete the UserForm_Initialize procedure as shown in following example. Setting the height of the dialog box to 120 points whei initially displayed specifies that the control at the bottom of the dialog bo›

be hidden when the dialog box opens.

```
Private Sub UserForm_Initialize()
    Me.Height = 120
End Sub
```

4. In the **Object** box in the **Code** window, click **CommandButton1**, and the select **Click** in the **Procedure** box. Complete the Image1_Click procedure shown in the following example. The example toggles the value of the **Hei** property of the UserForm between 120 points (the initial value) and 180 points.

```
Private Sub OptionButton1_Click()
    With Me
        .Height = 300 - .Height
    End With
End Sub
```

5. Run the dialog box. To hide or display the bottom section of the dialog bo: that contains the check box, click the command button. Click the **Close** button on the title bar to return to design mode.

Browsing Data with a TabStrip Control

You can use a **TabStrip** control to view different sets of information in the same of controls in a dialog box. For example, if you want to use one area of a dialog to display contact information pertaining to a group of individuals, you can creat **TabStrip** control and then add controls to contain the name, address, and phon number of each person in the group. You can then add a "tab" to the **TabStrip** control for each member of the group. After doing this, you can write code that, when you click a particular tab, updates the controls to display data about the person identified on that tab.

Tip To add, remove, rename, or move a tab in a tab strip, rightclick the tab st in design mode, and then click an item on the shortcut menu.

The following example changes the value of TextBox1 each time a different tab TabStrip1 is clicked. The index number of the tab that was clicked is passed to t event procedure.

```
Private Sub TabStrip1_Click(ByVal Index As Long)
If Index = 0 Then
    Me.TextBox1.Text = "7710 Betty Jane Lane"
ElseIf
Index = 1 Then
    Me.TextBox1.Text = "9523 15th Ave NE"
End If
End Sub
```

Keep in mind that formsrelated collections are 0based, which means that the in of the first member in any collection is 0 (zero).

Note Don't confuse TabStrip controls with MultiPage controls. Unlike a TabStri control, the pages (or tabs) of a MultiPage control contain a unique set of contro that you add during design time to each page. For information about using

MultiPage controls, see "Creating Tabs in Dialog Boxes" earlier in this chapter.

Data Validation

There are times when you'll want to make sure that the user only enters a valu a specific type in a particular control. For example, if you're using a **TextBox** control, which allows the user to enter any data type, and if your code expects t get a value of type **Integer** back from the text box, you should write code that verifies that the user has entered a valid integer before the dialog box closes. T verify that the user has entered the appropriate type of data in a control, you c check the value of the control either when the control loses the focus or when t dialog box is closed. The following example will prevent the user from moving t focus away from the txtCustAge text box without first entering a valid number.

```
Private Sub txtCustAge_Exit(ByVal Cancel As MSForms.ReturnBoolean)
    If Not IsNumeric(txtCustAge.Text) Then
       MsgBox "Please enter numeric value for the customer's age."
       Cancel = True
    End If
End Sub
```

Notice that you set the *Cancel* argument of a control's Exit event procedure to T to prevent the control from losing the focus.

To verify data before a dialog box closes, include code to check the contents of or more controls in the dialog box in the same routine that unloads the dialog b If a control contains invalid data, use an **Exit Sub** statement to exit the proced before the **Unload** statement can be executed. The following example runs whenever the user clicks the cmdOK command button.This procedure prevents user from closing the dialog box by using the cmdOK button until the txtCustAg text box contains a number.

```
Private Sub cmdOK_Click()
    If Not IsNumeric(txtCustAge.Text) Then
       MsgBox "Please enter numeric value for the customer's age."
       txtCustAge.SetFocus
       Exit Sub
    End If
    custAge = txtCustAge.Text
    Unload Me
End Sub
```

Getting Values When the Dialog Box Closes

Any data that a user enters in a dialog box is lost when the dialog box is closed. you return the values of controls in a UserForm after the form has been unload you get the initial values for the controls rather than any values the user may h entered.

If you want to save the data entered in a dialog box by a user, you can do so b saving the information to modulelevel variables while the dialog box is still runn The following example displays a dialog box and saves the data that's been ent in it.

```
'Code in module to declare public variables
```

```
Public strRegion As String
Public intSalesPersonID As Integer
Public blnCancelled As Boolean

'Code in form
Private Sub cmdCancel_Click()
          Module1.blnCancelled = True
          Unload Me
End Sub

Private Sub cmdOK_Click()
          'Save data
          intSalesPersonID = txtSalesPersonID.Text
          strRegion = lstRegions.List(lstRegions.ListIndex)
          Module1.blnCancelled = False
          Unload Me
End Sub

Private Sub UserForm_Initialize()
          Module1.blnCancelled = True
End Sub

'Code in module to display form
Sub LaunchSalesPersonForm()
          frmSalesPeople.Show
          If blnCancelled = True Then
                    MsgBox "Operation Cancelled!", vbExclamation
          Else

                    MsgBox "The Salesperson's ID is: " &
                              intSalesPersonID & _
                              "The Region is: " & strRegion
          End If
End Sub
```

Closing a Custom Dialog Box

Dialog boxes are always displayed as *modal*. That is, the user must close the di
box before returning to the document. Use the **Unload** statement to unload a
UserForm when the user indicates that he or she wants to close the dialog box.
Typically, you provide a command button in the dialog box that the user can clic
close it.

The following example inserts the name of a dialog box into a Word document a
then unloads the form. The code appears in the Click event for an **OK** button in
dialog box.

```
Private Sub cmdOK_Click()
    ActiveDocument.Content.InsertAfter txtUserName.Text
    Unload UserForm1
End Sub
```

Using the Same Dialog Box in Different Applications

Microsoft **Excel**, Word, and PowerPoint share features for creating custom dialo
boxes. You can create a UserForm in one of these applications and share it with
other applications.

▸To share a UserForm with another application

1. In the Visual Basic Editor for the application in which you created the UserForm, rightclick the UserForm in the **Project Explorer**, and then clicl

Export File on the shortcut menu.

2. Choose a name to export the UserForm as, and then click **Save**. The UserForm is saved with the .frm file name extension.

3. In the Visual Basic Editor for the application in which you want to use the UserForm, rightclick the project where you want to store the form in the Project Explorer, and then click **Import File** on the shortcut menu.

4. Select the name you gave the dialog box when you saved it, and then clic **Open**.

Note Not every UserForm that runs as it's supposed to in one application will r correctly when it's imported into another application. For example, if you import UserForm that contains Wordspecific code into Microsoft **Excel**, the UserForm w run correctly.

Working with Controls on a Document, Sheet, or Slide

Just as you can add **ActiveX** controls to custom dialog boxes, you can add cont directly to a document, sheet, or slide to make it interactive. For example, you might add text boxes, list boxes, option buttons, and other controls to a docum to turn it into an online form; you might add a button to a sheet that runs a commonly used macro; or you might add buttons and other controls to the slide a presentation to help the user navigate the slide show.

Although working with a control on a document, sheet, or slide is very similar tc working with a control in a custom dialog box, there are a few differences. Amo those differences are the following:

- On a document, sheet, or slide, you add controls by using the **Control Toolbox**, not the **Toolbox**. To display the **Control Toolbox**, point to **Toolbars** on the **View** menu, and then click **Control Toolbox**.

- When you're designing a custom dialog box, you run a dialog box to switc run mode, where your controls will respond to events, and you close a dia box and return to the Visual Basic Editor to switch back to design mode, where you can work with the controls without having them respond to eve When you're working with controls on documents or in workbooks, you cli the **Exit Design Mode** button on the **Visual Basic** toolbar to switch to ru mode, and you click the **Design Mode** button to switch back to design m In PowerPoint, you run a slide show to switch to run mode, and you switcl any editing view to switch back to design mode.

- A control may not have the same set of events on a document, sheet, or s as it does on a UserForm. For example, a command button on a UserForr an Exit event, whereas a command button on a document doesn't.

when the button is clicked.

Keep the following points in mind when you're working with controls on sheets:

- In addition to the standard properties available for **ActiveX** controls, you use the following properties with **ActiveX** controls in Microsoft **Excel**: **BottomRightCell**, **LinkedCell**, **ListFillRange**, **Placement**, **PrintObject** **TopLeftCell**, and **ZOrder**.

  You can set and return these properties by using the **ActiveX** control nam The following example scrolls through the workbook window until CommandButton1 is in the upperleft corner of the window.

```
Set t = Sheet1.CommandButton1.TopLeftCell
With ActiveWindow
          .ScrollRow = t.Row
          .ScrollColumn = t.Column
End With
```

- Some Microsoft **Excel** Visual Basic methods and properties are disabled w an **ActiveX** control is activated. For instance, you cannot use the **Sort** method when a control is active; thus, the following example will fail in a event procedure (because the control is still active after the user clicks it)

```
Private Sub CommandButton1_Click
          Range("a1:a10").Sort Key1:=Range("a1")
End Sub
```

  You can work around this problem by activating some other element on th sheet before you use the property or method that failed. For instance, the following example sorts the range.

```
Private Sub CommandButton1_Click
          Range("a1").Activate
          Range("a1:a10").Sort Key1:=Range("a1")
          CommandButton1.Activate
End Sub
```

- Controls in a Microsoft **Excel** workbook embedded in a document in anoth application won't work if the user doubleclicks the workbook to edit it. The controls will work if the user rightclicks the workbook and then clicks the **Open** command on the shortcut menu.

- When you save a Microsoft **Excel** 97 workbook by using the Microsoft **Exc** 5.0/95 Workbook file format, all **ActiveX** control information is lost.

- The **Me** keyword in an event procedure for an **ActiveX** control on a sheet refers to the sheet, not to the control.

Using ActiveX Controls on PowerPoint Slides

Adding controls to your PowerPoint slides provides a sophisticated way for you t

Using ActiveX Controls on Word Documents

You can add controls to documents to create interactive documents, such as onl forms. Keep the following points in mind when you're working with controls on documents:

- You can add **ActiveX** controls to either the text layer or the drawing layer add a control to the text layer, hold down the SHIFT key while you click th control on the **Control Toolbox** toolbar that you want to add to the document. To add a control to the drawing layer, click the control on the **Control Toolbox** toolbar without holding down the SHIFT key.

- A control you add to the text layer is an **InlineShape** object to which you gain access programmatically through the **InlineShapes** collection. A con you add to the drawing layer is a **Shape** object to which you gain access programmatically through the **Shapes** collection.

- Controls in the text layer are treated like characters and are positioned as characters within a line of text. For example, if you place controls in the c within a table, the controls will be automatically moved when you resize a columns in the table.

- You cannot drag controls from the **Control Toolbox** onto a Word docume When you press SHIFT and click a control to add it to the text layer, the control is automatically added at the insertion point. When you click a con to add it to the drawing layer, the position of the control is based on the position of the insertion point, but may not match it. If you add multiple controls to the drawing layer without moving the insertion point, the contı will all be placed in the same position, one on top of the other, so that yoı only see the last one you added.

- In design mode, **ActiveX** controls in the drawing layer are visible only in | layout view or online layout view.

- **ActiveX** controls in the drawing layer are always in run mode (so that the can receive input from a user) in page layout view or online layout view.

- If you want the user to be able to move between controls in an online forı pressing TAB, add the controls to the text layer, and protect the form by clicking the **Protect Form** button on the **Forms** toolbar.

- If you want to add form fields instead of **ActiveX** controls to your docume to create an online form, use the **Forms** toolbar.

Using ActiveX Controls on Microsoft Excel Sheets

You can add controls to worksheets or chart sheets next to the data the control: linked to so that they're easy for the user to find and understand, and so that u: them causes only minimal interruptions during a work session. For example, yoı can add to a worksheet a button that runs a procedure that formats the active c exchange information with the user while a slide show is running. For example, can use controls on slides so that viewers of a show designed to be run in a kios have a way to choose options and then run a custom show based on the viewer choices.

Keep the following points in mind when you're working with controls on slides:

- A control on a slide is in design mode except when the slide show is runni
- If you want a particular control to appear on all the slides in a presentatio add the control to the slide master.
- The **Me** keyword in an event procedure for a control on a slide refers to th slide. The **Me** keyword in an event procedure for a control on a master ret to the master, not to the slide that's being displayed when the control eve is triggered.
- Writing event code for controls on slides is very similar to writing event cc for controls in dialog boxes. The following example (the Click event procec for the command button named "cmdChangeColor") sets the background the slide the button is on.

```
Private Sub cmdChangeColor_Click()
   With Me
        .FollowMasterBackground = Not .FollowMasterBackground
        .Background.Fill.PresetGradient msoGradientHorizontal, 1, msoGradientBrass
    End With
End Sub
```

- You may want to use controls to provide your slide show with navigation t that are more complex than those built into PowerPoint. For instance, if yc add two buttons named "cmdBack" and "cmdForward" to the slide master write the code in the following example for them, all slides based on the master (and set to show master background graphics) will have these professionallooking navigation buttons, which will be active during a slide show.

```
Private Sub cmdBack_Click()
    Me.Parent.SlideShowWindow.View.Previous
End Sub

Sub cmdForward_Click()
    Me.Parent.SlideShowWindow.View.Next
End Sub
```

- To work with all the **ActiveX** controls on a particular slide without affectin the other shapes on the slide, you can construct a **ShapeRange** collectioı that contains only controls. You can then either apply properties and meth to the entire collection or iterate through the collection to work with each control individually. The following example aligns all the controls on slide in the active presentation and arranges them vertically.

```
With ActivePresentation.Slides(1).Shapes
    numShapes = .Count
    If numShapes > 1 Then
       numControls = 0
       ReDim ctrlArray(1 To numShapes)
       For i = 1 To numShapes
            If .Item(i).Type = msoOLEControlObject Then
                numControls = numControls + 1
                ctrlArray(numControls) = .Item(i).Name
             End If
       Next
       If numControls > 1 Then
             ReDim Preserve ctrlArray(1 To numControls)
             Set ctrlRange = .Range(ctrlArray)
             ctrlRange.Distribute msoDistributeVertically, True
             ctrlRange.Align msoAlignLefts, True
       End If
    End If
End With
```

Working with Controls Programmatically

To gain access to a control programmatically, you can either refer to the control its code name or get to it through the collection it belongs to. (The code name c control is the value of the **(Name)** property for that control in the **Properties** window.)

The following example sets the caption for the control named "CommandButton

```
CommandButton1.Caption = "Run"
```

Note that when you use a control name outside the class module for the docum sheet, or slide that contains the control, you must qualify the control name with code name of the document, sheet, or slide. The following example changes the caption on the control named "CommandButton1" on the Sheet1.

```
Sheet1.CommandButton1.Caption = "Run"
```

You can also gain access to **ActiveX** controls through the **Shapes**, **OLEObjects** **InlineShapes** collection. **ActiveX** controls you add to the drawing layer of a document, sheet, or slide are contained in **Shape** objects and can be programmatically controlled through the **Shapes** collection. In Microsoft **Excel**, **ActiveX** controls are also contained in **OLEObject** objects that can be controlle through the **OLEObjects** collection. In Word, **ActiveX** controls you add to the t layer of a document are contained in **InlineShape** objects and can be controlle through the **In**lineShapes collection.

Important You use the name of the **Shape** object that contains a particular control, not the code name of the control, to gain access to the control programmatically through a collection. In Microsoft **Excel** and PowerPoint, the name of the object that contains a control matches the code name of the contrc default. This isn't true in Word, however; the name of the object that contains a control (which will be something like "Control 1" by default) is unrelated to the name of a control (which will be something like "CommandButton1" by default). change the code name of a control, select the control and change the value to t right of **(Name)** in the **Properties** window. To change the name of the **Shape** object, **OLEObject** object, or other object that contains the control, change the value of its **Name** property.

The following example adds a command button to worksheet one.

```
Worksheets(1).OLEObjects.Add "Forms.CommandButton.1", _
          Left:=10, Top:=10, Height:=20, Width:=100
```

The following example sets the **Left** property for CommandButton1 on workshe
one.

```
Worksheets(1).OLEObjects("CommandButton1").Left = 10
```

The following example sets the caption for CommandButton1.

```
Worksheets(1).OLEObjects("CommandButton1").Object.Caption = "Run"
```

The following example adds a check box to the active document's text layer.

```
ActiveDocument.InlineShapes.AddOLEControl ClassType:="Forms.CheckBox.1"
```

The following example sets the **Width** property for the first shape in the active document's text layer.

```
ActiveDocument.InlineShapes(1).Width = 200
```

The following example sets the **Value** property for the first shape the active document's text layer.

```
ActiveDocument.InlineShapes(1).OLEFormat.Object.Value = True
```

The following example adds a combo box to the active document's drawing layer.

```
ActiveDocument.Shapes.AddOLEControl ClassType:="Forms.ComboBox.1"
```

The following example sets the **Left** property for a combo box contained in Control 1 in the active document's drawing layer.

```
ActiveDocument.Shapes("Control 1").Left = 100
```

The following example sets the **Text** property for a combo box contained in Control 1 in the active document's drawing layer.

```
ActiveDocument.Shapes("Control 1").OLEFormat.Object.Text = "Reed"
```

The following example adds a command button to slide one in the active presentation.

```
ActivePresentation.Slides(1).Shapes.AddOLEObject Left:=100, Top:=10
    Width:=150, Height:=50, ClassName:="Forms.CommandButton.1"
```

The following example sets the **Left** property for the control contained in CommandButton1 on slide one in the active presentation.

```
ActivePresentation.Slides(1).Shapes("CommandButton1").Left = 100
```

The following example sets the **Caption** property for the control contained in CommandButton1 on slide one in the active presentation.

```
ActivePresentation.Slides(1).Shapes("CommandButton1") _
    .OLEFormat.Object.Caption = "Run"
```